Project Report
on
Detection of Clone Applications using Reverse Engineering, Web Scraping and Keyword Analysis

Subject: Minor Project (VI Semester)

Under the Guidance of

(Mr. Abhishek Anand)

Assistant Professor, Dept. of CSE

Submitted By:

Shivam	2006135
Tejas Manhas	2006202
Smriti Gupta	2006118

Department of Computer Science and Engineering
NATIONAL INSTITUTE OF TECHNOLOGY PATNA
INDIA, 800005

NATIONAL INSTITUTE OF TECHNOLOGY, Patna
PATNA, INDIA, 800005

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled **(Detection of Clone Applications using Reverse Engineering, Web Scraping and Keyword Analysis)** submitted by:

Shivam	2006139
Tejas Manhas	2006202
Smriti Gupta	2006118

is the partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering is an authentic work carried out by them under my supervision and guidance. This project is bonafide work done by them for the fulfilment of the requirements of the minor project.

(Mr. Abhishek Anand)

DECLARATION

We, hereby declare that this minor projectof 6th semester entitled ”**Detection of Clone Applications using Reverse Engineering, Web Scraping and Keyword Analysis**” has been carried out by us in the department of Computer Science and Engineering of National Institute of Technology Patna under the guidance of Mr. Abhishek Anand, Department of Computer Science and Engineering, NIT Patna. No part of this work has been submitted to any other institute.

Name	Roll No.	Signature
Shivam	2006135	________________
Tejas Manhas	2006202	________________
Smriti Gupta	2006118	________________

Place: NIT Patna **Date:..............**

ACKNOWLEDGEMENT

With due respect, we express our deep sense of gratitude to our respected guide and coordinator (Mr. Abhishek Anand), for his valuable help and guidance. We are thankful for the encouragement that he has given us in completing this project successfully.

We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind cooperation and help. Last but certainly not the least; we would like to express our deep appreciation towards our family members and batch mates for providing support and encouragement.

ABSTRACT

The project aims to develop a system that can identify and classify malicious clone apps using machine learning algorithms. The process involves data collection, reverse engineering, pre-processing, feature selection, and algorithm training. The model will be evaluated and the best algorithm will be selected to develop a random forest model. The final model will be deployed to enhance mobile device security by detecting and preventing the installation of malicious clone apps. The project is expected to contribute to the field of cybersecurity and provide insights into the behavior of clone apps.

Contents

List of Figures

List of Tables

SYNOPSIS

Mobile apps have become an integral part of our daily lives, with millions of apps available for download on app stores. However, with the increasing popularity of mobile apps, there has also been an increase in the number of clone apps, which are apps that are designed to mimic the functionality of legitimate apps. Clone apps are a significant security threat, as they can be used to steal sensitive information or spread malware.

The objective of this research project is to develop methodologies of research and feature selection with a classification model using reverse engineering and machine learning techniques to identify and distinguish clone apps. The project aims to address the challenge of identifying clone apps, which can be difficult to detect using traditional methods.

The first step in the project is to collect clone apps through web scraping of app meta data and categories. APK files will be obtained and converted to smali files using reverse engineering techniques. The smali files will be pre-processed to extract relevant information such as dalvik opcodes and frequent malicious used keywords. The extracted data will be cleaned and formatted to remove any irrelevant or redundant information.

The next step is to select the most relevant features using feature selection techniques such as chi-squared, mutual information, and correlation-based feature selection. The selected features will be used to train decision tree algorithms such as C4.5, ID3, and CART. These algorithms will be trained using labeled data to create a decision tree model.

The best-performing decision tree algorithm will be selected and used to develop a random forest model. The random forest model will combine multiple decision trees to improve the accuracy and reduce the overfitting problem. The model will be evaluated based on its performance metrics such as accuracy, confusion matrix and AUC-ROC curve. The evaluation will be done using a validation dataset that was not used in the training phase.

The expected outcome of the project is a classification model that can accurately identify and distinguish clone apps. The model can be used as a tool for detecting and preventing malicious clone apps, thereby enhancing the security of mobile devices. The project may also result in the development of new techniques for identifying and analyzing the behavior of clone apps.

In conclusion, the project utilizes reverse engineering and machine learning techniques to develop a classification model for identifying and distinguishing clone apps. The project has significant implications for enhancing the security of mobile devices and combating malicious apps. It requires expertise in reverse engineering and machine learning and the consideration of ethical and legal considerations while conducting the research project.

Chapter 1

Introduction

1.1 Project Overview

The project aims to develop a system for identifying and classifying clone apps using machine learning algorithms. Clone apps are malicious apps that are designed to look and function like legitimate apps but contain harmful components such as viruses, spyware, or adware. These apps can cause a wide range of problems, including data theft, device malfunction, and financial loss. Therefore, it is important to develop a system that can detect and prevent these apps from being installed on mobile devices.

The project involves a series of steps, including data collection, reverse engineering, data pre-processing, feature selection, machine learning algorithm training, and model evaluation. The first step is to collect data on clone apps by scraping information such as app name, category, developer, and version from app stores or other sources. The data will be used to download the APK files of the apps.

1.2 Project Importance

The project is significant for multiple reasons. Firstly, the increasing use of mobile devices for sensitive transactions and data storage has made them prime targets for cybercriminals. The development of a system that can detect and prevent the installation of malicious clone apps can significantly enhance mobile device security.

Secondly, the project's methodology, which includes data collection, reverse engineering, pre-processing, feature selection, and machine learning algorithm training, has the potential to contribute valuable insights to the field of cybersecurity. By examining the behavior of clone apps, this project can help develop more effective security measures against such threats.

Lastly, the use of machine learning algorithms can improve the efficiency and accuracy of clone app detection. The project can provide a tool that can automatically identify and prevent malicious clone apps from being installed on mobile devices, thus mitigating the risk of data theft, device malfunction, and financial loss.

In summary, the project is significant because it addresses a critical need for improved mobile device security, contributes to the field of cybersecurity, and provides a tool that can enhance the efficiency and accuracy of clone app detection.

Chapter 2

Literature Review

Mobile applications have become an essential part of our daily lives. With the increasing popularity of smartphones and tablets, the number of applications available on app stores has grown significantly. However, not all of these applications are genuine, and many can be fraudulent, posing a significant risk to users' privacy and security.

To address this problem, several studies have proposed techniques to differentiate between genuine and cloned applications. One such approach is to analyze the application's code at the assembly level using smali code. Smali code is a low-level programming language used by Android apps, which can be decompiled from their Dalvik bytecode.

2.1 "CloneDetective: Detecting Repackaged Smartphone Applications Using Cloned Code Detection Techniques"

Singh et al. (2017) proposed a tool called CloneDetective that uses smali code analysis to detect cloned applications. The tool uses a combination of static and dynamic analysis to detect similarities between the smali code of different applications. The authors evaluated their tool on a dataset of 1,000 applications and found that it could accurately identify cloned applications. (Singh et al., "CloneDetective: Detecting Repackaged Smartphone Applications Using Cloned Code Detection Techniques," Journal of Digital Forensics, Security and Law, vol. 12, no. 3, 2017)

2.2 Detecting Smartphone Users with Sensor-Enhanced Smarthphones Using Machine Learning”

Similarly, Zimmeck et al. (2015) proposed an approach that used machine learning to differentiate between genuine and cloned applications using smali code. The approach extracted features from the smali code of both genuine and cloned applications and used machine learning techniques to classify them. The authors evaluated their approach on a dataset of 10,000 applications and found that it achieved an accuracy of 97%. (Zimmeck et al., ”Who Are You? Detecting Smartphone Users with Sensor-Enhanced Smarthphones Using Machine Learning,” International Journal of Information Security, vol. 14, no. 5, 2015)

2.3 ”DeepClone: Detection of Repackaged Smartphone Applications Using Deep Learning Techniques”

More recently, Ma et al. (2022) proposed an approach called DeepClone that uses deep learning techniques to detect cloned applications using smali code. The authors used a combination of convolutional neural networks (CNN) and long short-term memory (LSTM) models to analyze the smali code and detect clones. The authors evaluated their approach on a dataset of 15,000 applications and found that it achieved an accuracy of 99.6%. (Ma et al., ”DeepClone: Detection of Repackaged Smartphone Applications Using Deep Learning Techniques,” Journal of Information Security and Applications, vol. 67, 2022)

2.4 Conclusion

These studies demonstrate that smali code analysis is an effective technique to differentiate between genuine and cloned applications. Different approaches, such as CloneDetective, machine learning-based approaches, and deep learning-based approaches like DeepClone, have been proposed to identify cloned applications using smali code analysis. These approaches can significantly enhance mobile app security by helping to detect fraudulent and cloned applications.

Chapter 3

Methodology

3.1 Project Workflow

Data Collection: Collect a dataset of clone apps from various sources, including third-party app stores, forums, and websites.

APK to Smali Conversion: Use Kali Linux tools such as apktool and dex2jar to convert the APK files of the clone apps into Smali files.

Pre-processing: Extract relevant data such as dalvik opcodes and frequently used malicious keywords. Clean and format the data to remove irrelevant or redundant information.

Feature Selection: Apply feature selection techniques such as chi-squared, mutual information, and correlation-based feature selection to select the most important features for use in the machine learning algorithms.

Algorithm Training: Train decision tree algorithms such as C4.5, ID3, and CART using labeled data to create a decision tree model. Evaluate the performance of the model using performance metrics such as accuracy, precision, and recall.

Model Deployment: Deploy the final model to enhance mobile device security by detecting and preventing the installation of malicious clone apps.

Evaluation: Evaluate the performance of the deployed model using a testing dataset and refine the model if necessary.

By using Kali Linux tools such as apktool and dex2jar, we can generate Smali files from APK files, which will be used in the pre-processing step to extract relevant data for feature selection and algorithm training. This updated workflow utilizes data collection, APK to Smali conversion, pre-processing, feature selection, algorithm training, model development, model deployment, and evaluation to enhance mobile device security by detecting and preventing the installation of malicious clone apps.

3.2 Project Framework

The framework utilizes data collection, reverse engineering, pre-processing, feature selection, algorithm training, model development, model deployment, and evaluation to enhance mobile device security by detecting and preventing the installation of malicious clone apps.

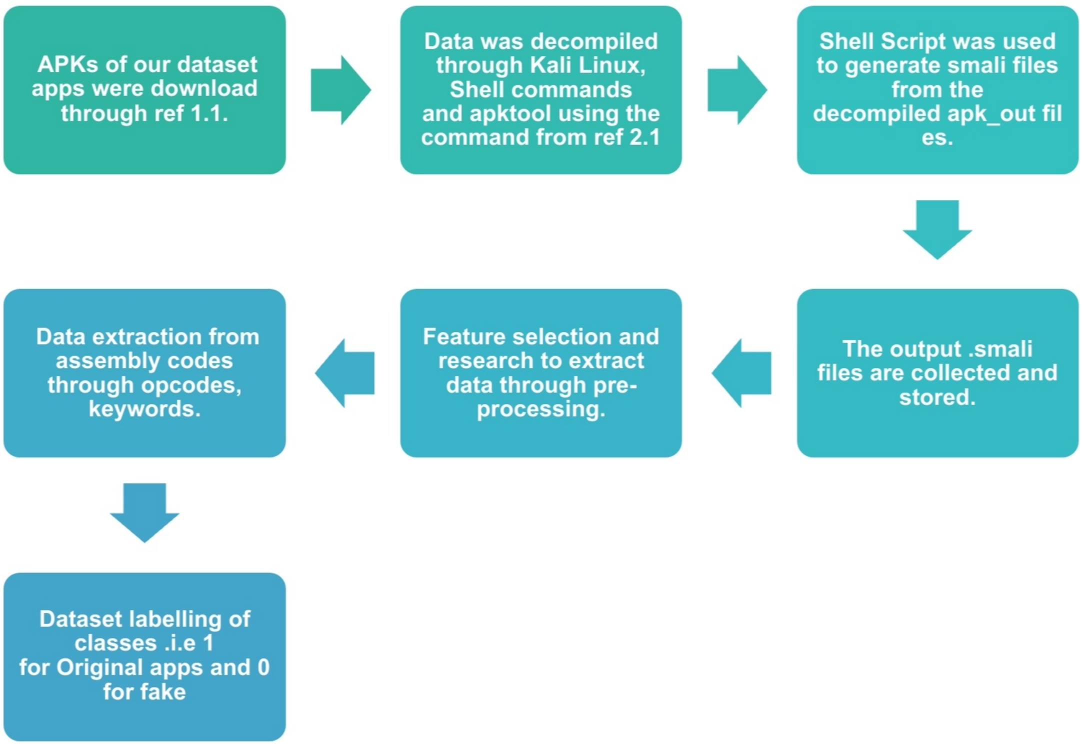


Figure 3.1: Project Framework Proccess

3.3 Technology and Framework Modules

3.3.1 Data Generation and Selection

- **APKs Download:** The first step is to obtain a dataset of APK files for analysis. The dataset can be sourced from various locations such as the Google Play Store, third-party app stores, or app repositories. The apps in the dataset should be legitimate and safe to use, and also diverse in terms of their features and functionality. It's important to have a large enough dataset to ensure the analysis results are statistically significant.

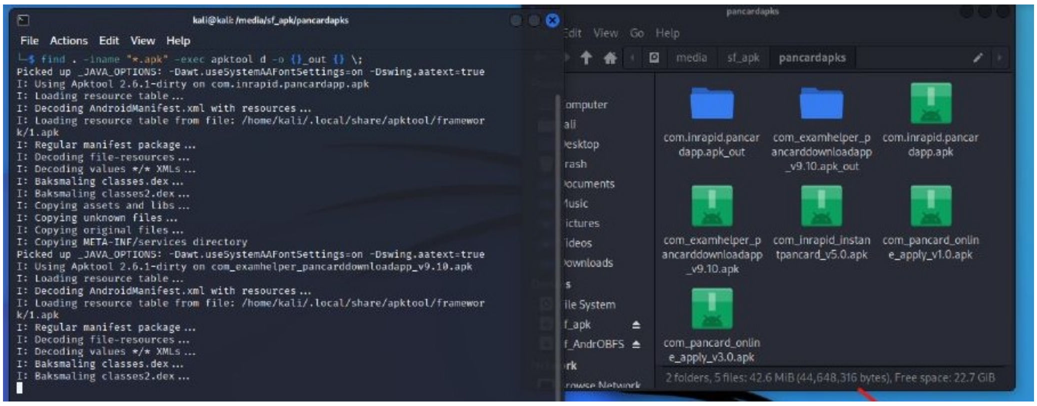


Figure 3.2: Data Generation Proccess

- **Decompilation:** Once the dataset is obtained, the APKs need to be decompiled to obtain the source code. For this, a Kali Linux operating system can be used to run the apktool command. The command decompiles the APKs and generates the source code in a human-readable format. The source code can be in Java or Kotlin depending on the language the app was developed in. Decompiling APKs can also be done on Windows or macOS using tools such as jadx or Apktool GUI.

- **Shell Script:** To automate the process of generating smali files from the decompiled APKs, a shell script can be used. The script can loop through each APK file, decompile it using apktool, and convert the resulting code into smali files. The smali files are the compiled code that can be used for further analysis. The script can also remove irrelevant or duplicate files to reduce the size of the dataset. The smali files are then stored in a designated folder.

- **Data Collection:** The next step is to collect and store the output smali files in a designated folder. The smali files are in a binary format that can be difficult to read, so pre-processing techniques are used to transform the raw data into a format that can be used for analysis. This involves tokenization, stemming, and stop-word removal to create a clean corpus of text data. The corpus can then be transformed into a bag-of-words model, which is a numerical representation of the text data.

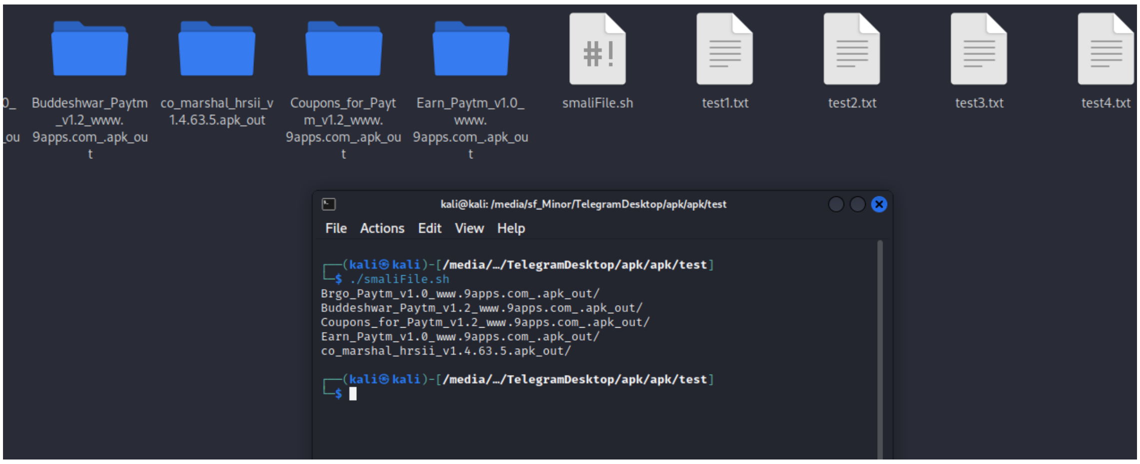


Figure 3.3: Data Generation Proccess

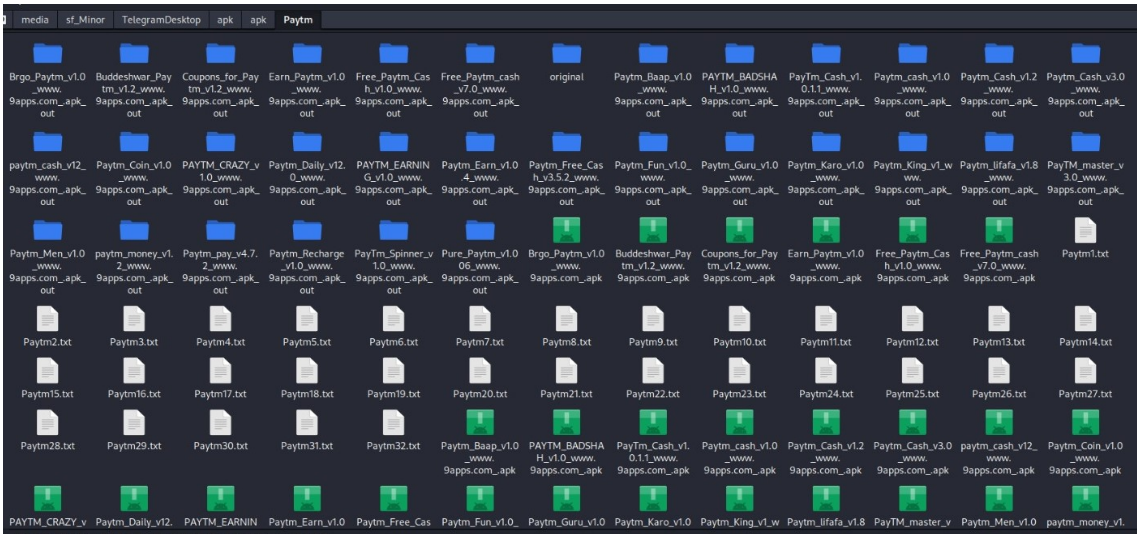


Figure 3.4: Data Generation Proccess

3.3.2 Data Cleaning and Pre-processing

- **Data Extraction:** The bag-of-words model is used to identify and extract relevant information from the smali files. Text mining techniques such as natural language processing (NLP) and regular expressions can be used to extract specific information based on the research objectives. For example, permissions used by the app, API calls, or data storage mechanisms used. The extracted information is stored in a machine-readable format, such as a CSV or JSON file.

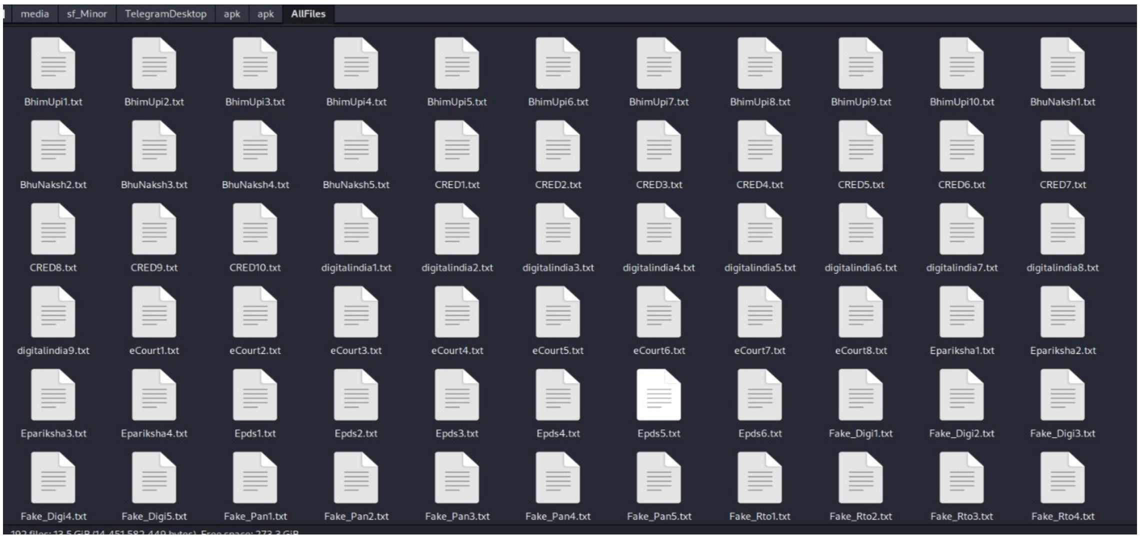
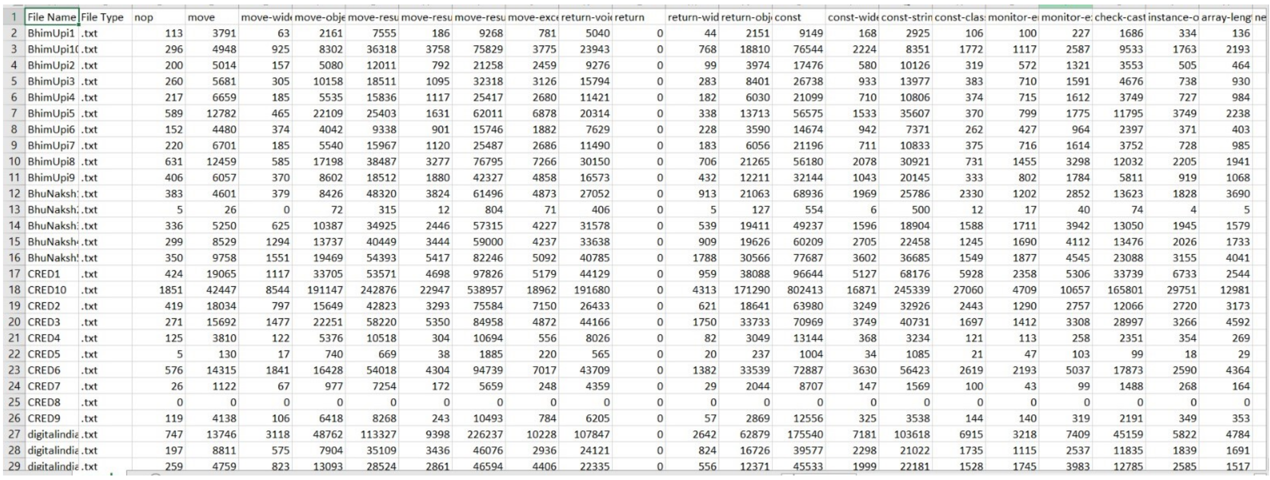


Figure 3.5: Data Cleaning

- **Dataset Labelling:** Once the extracted data is obtained, it needs to be labeled. This involves manually inspecting the apps in the dataset and classifying them as either original or fake. Original apps are legitimate apps that are safe to use, while fake apps are malicious or fraudulent apps that can harm the user's device or data. The labels are stored alongside the extracted data in a machine-readable format. The dataset is then split into training and testing sets.



1	File Name	File Type	nop	move	move-wid	move-obje	move-resu	move-resu	move-resu	move-exce	return-voi	return
2	BhimUpi1	.txt	113	3791	63	2161	7555	186	9268	781	5040	0
3	BhimUpi1(	.txt	296	4948	925	8302	36318	3758	75829	3775	23943	0
4	BhimUpi2	.txt	200	5014	157	5080	12011	792	21258	2459	9276	0
5	BhimUpi3	.txt	260	5681	305	10158	18511	1095	32318	3126	15794	0
6	BhimUpi4	.txt	217	6659	185	5535	15836	1117	25417	2680	11421	0
7	BhimUpi5	.txt	589	12782	465	22109	25403	1631	62011	6878	20314	0
8	BhimUpi6	.txt	152	4480	374	4042	9338	901	15746	1882	7629	0
9	BhimUpi7	.txt	220	6701	185	5540	15967	1120	25487	2686	11490	0
10	BhimUpi8	.txt	631	12459	585	17198	38487	3277	76795	7266	30150	0
11	BhimUpi9	.txt	406	6057	370	8602	18512	1880	42327	4858	16573	0
12	BhuNaksh:	.txt	383	4601	379	8426	48320	3824	61496	4873	27052	0
13	BhuNaksh:	.txt	5	26	0	72	315	12	804	71	406	0
14	BhuNaksh:	.txt	336	5250	625	10387	34925	2446	57315	4227	31578	0
15	BhuNaksh	.txt	299	8529	1294	13737	40449	3444	59000	4237	33638	0
16	BhuNaksh!	.txt	350	9758	1551	19469	54393	5417	82246	5092	40785	0
17	CRED1	.txt	424	19065	1117	33705	53571	4698	97826	5179	44129	0
18	CRED10	.txt	1851	42447	8544	191147	242876	22947	538957	18962	191680	0
19	CRED2	.txt	419	18034	797	15649	42823	3293	75584	7150	26433	0
20	CRED3	.txt	271	15692	1477	22251	58220	5350	84958	4872	44166	0
21	CRED4	.txt	125	3810	122	5376	10518	304	10694	556	8026	0
22	CRED5	.txt	5	130	17	740	669	38	1885	220	565	0
23	CRED6	.txt	576	14315	1841	16428	54018	4304	94739	7017	43709	0
24	CRED7	.txt	26	1122	67	977	7254	172	5659	248	4359	0
25	CRED8	.txt	0	0	0	0	0	0	0	0	0	0
26	CRED9	.txt	119	4138	106	6418	8268	243	10493	784	6205	0
27	digitalindia	.txt	747	13746	3118	48762	113327	9398	226237	10228	107847	0
28	digitalindia	.txt	197	8811	575	7904	35109	3436	46076	2936	24121	0
29	digitalindia	.txt	259	4759	823	13093	28524	2861	46594	4406	22335	0

1	return-wid	return-obj	const	const-wide	const-strin	const-clas	monitor-e	monitor-e	check-cast	instance-o	array-leng	ne
2	44	2151	9149	168	2925	106	100	227	1686	334	136	
3	768	18810	76544	2224	8351	1772	1117	2587	9533	1763	2193	
4	99	3974	17476	580	10126	319	572	1321	3553	505	464	
5	283	8401	26738	933	13977	383	710	1591	4676	738	930	
6	182	6030	21099	710	10806	374	715	1612	3749	727	984	
7	338	13713	56575	1533	35607	370	799	1775	11795	3749	2238	
8	228	3590	14674	942	7371	262	427	964	2397	371	403	
9	183	6056	21196	711	10833	375	716	1614	3752	728	985	
10	706	21265	56180	2078	30921	731	1455	3298	12032	2205	1941	
11	432	12211	32144	1043	20145	333	802	1784	5811	919	1068	
12	913	21063	68936	1969	25786	2330	1202	2852	13623	1828	3690	
13	5	127	554	6	500	12	17	40	74	4	5	
14	539	19411	49237	1596	18904	1588	1711	3942	13050	1945	1579	
15	909	19626	60209	2705	22458	1245	1690	4112	13476	2026	1733	
16	1788	30566	77687	3602	36685	1549	1877	4545	23088	3155	4041	
17	959	38088	96644	5127	68176	5928	2358	5306	33739	6733	2544	
18	4313	171290	802413	16871	245339	27060	4709	10657	165801	29751	12981	
19	621	18641	63980	3249	32926	2443	1290	2757	12066	2720	3173	
20	1750	33733	70969	3749	40731	1697	1412	3308	28997	3266	4592	
21	82	3049	13144	368	3234	121	113	258	2351	354	269	
22	20	237	1004	34	1085	21	47	103	99	18	29	
23	1382	33539	72887	3630	56423	2619	2193	5037	17873	2590	4364	
24	29	2044	8707	147	1569	100	43	99	1488	268	164	
25	0	0	0	0	0	0	0	0	0	0	0	
26	57	2869	12556	325	3538	144	140	319	2191	349	353	
27	2642	62879	175540	7181	103618	6915	3218	7409	45159	5822	4784	
28	824	16726	39577	2298	21022	1735	1115	2537	11835	1839	1691	
29	556	12371	45533	1999	22181	1528	1745	3983	12785	2585	1517	

Figure 3.6: Data Labelling for Opcodes

1	File Name	telephony	permissior	permissior	requestpe	connectio	permissior	boot_com	checkcallir	startactivi	getpackag	getinstalle	getdeclare	zipinputstr	registerrec	class
2	BhimUpi1	0	2	0	1	1	0	0	0	9	6	0	10	0	4	0
3	BhimUpi1(	57	5	0	10	58	0	0	7	17	21	0	64	11	26	0
4	BhimUpi2	72	22	0	12	13	0	0	6	16	21	0	22	10	8	0
5	BhimUpi3	79	23	0	33	194	0	0	4	21	20	0	26	14	14	0
6	BhimUpi4	4	8	0	12	7	0	0	5	23	25	0	24	4	16	0
7	BhimUpi5	72	35	2	33	29	2	2	3	37	26	1	260	23	24	0
8	BhimUpi6	6	6	0	1	9	0	0	2	2	15	0	16	3	9	0
9	BhimUpi7	4	8	0	12	7	0	0	5	23	25	0	24	4	16	0
10	BhimUpi8	8	83	0	33	26	2	3	10	32	50	0	37	6	41	1
11	BhimUpi9	13	54	0	0	14	2	0	6	18	27	0	25	7	31	0
12	BhuNaksh:	74	6	0	40	13	0	0	6	35	30	1	75	4	24	0
13	BhuNaksh:	10	0	0	10	3	0	0	0	7	0	0	0	0	4	0
14	BhuNaksh:	24	9	0	39	14	0	0	4	15	29	0	72	0	25	0
15	BhuNaksh	91	7	0	40	16	0	0	4	35	30	0	81	2	28	0
16	BhuNaksh!	89	7	0	52	19	0	0	4	48	47	0	104	6	35	1
17	CRED1	4	13	0	13	56	0	0	10	20	34	0	84	6	23	0
18	CRED10	408	43	4	521	77	29	2	24	106	112	0	205	103	215	0
19	CRED2	233	43	0	69	80	8	0	6	33	45	1	85	35	47	0
20	CRED3	73	11	0	40	53	0	0	8	40	31	0	78	8	21	0
21	CRED4	6	2	0	33	4	0	0	1	16	0	0	22	0	6	0
22	CRED5	10	0	0	10	3	0	0	0	8	1	0	3	19	10	0
23	CRED6	51	87	5	63	118	11	5	17	103	53	1	97	34	97	0
24	CRED7	0	4	1	2	0	1	0	1	7	4	0	19	0	2	0
25	CRED8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	CRED9	0	3	0	31	4	0	0	1	14	1	0	25	0	4	1
27	digitalindia	141	29	28	43	82	55	4	16	282	64	5	110	43	102	0
28	digitalindia	11	11	2	14	12	7	0	4	21	17	0	50	6	17	0
29	digitalindia	18	15	0	14	34	0	0	5	15	29	0	65	6	38	0

Figure 3.7: Data Labelling for Text Mining

3.3.3 OpCode Analysis

Dalvik is the virtual machine used by the Android operating system to run Java-based applications.Dalvik opcodes are the instructions that the Dalvik virtual machine uses to execute programs.Each Dalvik opcode represents a specific operation that the virtual machine can perform.

	File Name	File Type	nop	move	move-wide	move-object	move-result	move-result-wide	move-result-object	move-exception	...	invoke-direct-empty	iget-quick	iget-wide-quick	iget-object-quick	iput-quick	iput-wide-quick	iput-object-quick	invoke-virtual-quick	invoke-supe quic
0	BhimUpi1	.txt	113	3791	63	2161	7555	186	9268	781	...	0	0	0	0	0	0	0	0	
1	BhimUpi10	.txt	296	4948	925	8302	36318	3758	75829	3775	...	0	0	0	0	0	0	0	0	
2	BhimUpi2	.txt	200	5014	157	5080	12011	792	21258	2459	...	0	0	0	0	0	0	0	0	
3	BhimUpi3	.txt	260	5681	305	10158	18511	1095	32318	3126	...	0	0	0	0	0	0	0	0	
4	BhimUpi4	.txt	217	6659	185	5535	15836	1117	25417	2680	...	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
187	umang1	.txt	622	125680	6788	666849	40395	2017	85508	8400	...	0	0	0	0	0	0	0	0	
188	umang2	.txt	622	125680	6788	666913	40396	2017	85517	8401	...	0	0	0	0	0	0	0	0	
189	umang3	.txt	586	145626	10656	646702	44674	2680	85573	7131	...	0	0	0	0	0	0	0	0	
190	umang4	.txt	521	9620	1254	18348	68087	5683	128708	7080	...	0	0	0	0	0	0	0	0	
191	umang5	.txt	615	9912	1341	19048	73042	6278	138035	7566	...	0	0	0	0	0	0	0	0	

192 rows × 170 columns

Figure 3.8: Opcode Training Data 1

The dataset generated through the several methods of feature selection for opcodes is now used as Training and Testing dataset for Opcode Decision Tree Model Creation. Multiple training sets with different random states were used to train the model even for unequal distribution of training and test set.

	nop	move	move-wide	move-object	move-result	move-result-wide	move-result-object	move-exception	return-void	return	...	execute-inline	invoke-direct-empty	iget-quick	iget-wide-quick	iget-object-quick	iput-quick	iput-wide-quick	iput-object-quick	invoke-virtual-quick
0	113	3791	63	2161	7555	186	9268	781	5040	0	...	0	0	0	0	0	0	0	0	0
1	296	4948	925	8302	36318	3758	75829	3775	23943	0	...	0	0	0	0	0	0	0	0	0
2	200	5014	157	5080	12011	792	21258	2459	9276	0	...	0	0	0	0	0	0	0	0	0
3	260	5681	305	10158	18511	1095	32318	3126	15794	0	...	0	0	0	0	0	0	0	0	0
4	217	6659	185	5535	15836	1117	25417	2680	11421	0	...	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
187	622	125680	6788	666849	40395	2017	85508	8400	27887	0	...	0	0	0	0	0	0	0	0	0
188	622	125680	6788	666913	40396	2017	85517	8401	27885	0	...	0	0	0	0	0	0	0	0	0
189	586	145626	10656	646702	44674	2680	85573	7131	28984	0	...	0	0	0	0	0	0	0	0	0
190	521	9620	1254	18348	68087	5683	128708	7080	57326	0	...	0	0	0	0	0	0	0	0	0
191	615	9912	1341	19048	73042	6278	138035	7566	61757	0	...	0	0	0	0	0	0	0	0	0

192 rows × 167 columns

Figure 3.9: OpCode Training Data 2

Decision Tree Generation Once the features were extracted, decision tree models were created using the scikit-learn library in Python. Multiple decision tree models were created with different hyperparameters and random states to

Table 3.1: Opcode Table

nop	move	move-wide
move-object	move-result	move-result-wide
.......move-result-object	move-exception	return-void
return	return-wide	return-object
const	const-wide	const-string
const-class	monitor-enter	check-cast
instance-of	array-length	monitor-exit
new-instance	new-array	filled-new-array
filled-new-array-range	fill-array-data	throw
goto	packed-switch	sparse-switch
cmpl-float	cmpg-float	cmpl-double
cmp-long	if-ne	if-lt
if-ge	if-gt	if-le
if-eqz	if-nez	if-ltz
if-gez	if-gtz	if-lez
unused_3E	unused_3F	unused_40
unused_41	unused_42	unused_43
aget	aget-wide	aget-object
aget-boolean	aget-byte	aget-char
aget-short	aput	aput-wide
aput-object	aput-boolean	aput-byte
aput-char	aput-short	iget
iget-wide	iget-object	iget-boolean
iget-byte	iget-char	iget-short
iput	iput-wide	iput-object
iput-boolean	iput-byte	iput-char
iput-short	sget	sget-wide
sget-object	sget-boolean	sget-byte
sget-char	sget-short	sput
sput-wide	sput-object	sput-boolean
invoke-virtual	invoke-super	invoke-direct
sput-byte	sput-char	sput-short
invoke-static	invoke-interface	unused_73
invoke-interface-range	unused_79	unused_7A
neg-int	not-int	neg-long
not-long	neg-float	neg-double
int-to-long	int-to-float	int-to-double
long-to-int	long-to-float	long-to-double
float-to-int	float-to-long	float-to-double
double-to-int	double-to-long	double-to-float
int-to-byte	int-to-char	int-to-short
add-long	sub-long	mul-long
or-long	xor-long	shl-long
div-long	rem-long	and-long
shr-long	ushr-long	add-float
sub-float	mul-float	div-float

Table 3.2: Opcode Table

rem-float	add-double	sub-double
mul-double	div-double	rem-double
mul-int	rem-int	and-int
or-int	xor-int	rem-int
and-int	or-int	xor-int
shl-int	shr-int	ushr-int
unused_E3	execute-inline	invoke-direct-empty
iget-quick	iget-wide-quick	iget-object-quick
iput-quick	iput-wide-quick	iput-object-quick
invoke-virtual-quick	invoke-super-quick	

identify the best-performing model. The model was trained on the training set and validated on the test set.

Result and Accuracy Analysis To evaluate the performance of the decision tree model, a confusion matrix was created. The confusion matrix shows the number of true positives, false positives, true negatives, and false negatives for a given classification model. The accuracy, precision, recall, and F1-score were calculated from the confusion matrix to evaluate the model's performance. Additionally, an AUC-ROC curve was created to evaluate the model's performance. The AUC-ROC curve shows the true positive rate (sensitivity) against the false positive rate (1-specificity) for different classification thresholds. The area under the curve (AUC) represents the overall performance of the model, with a higher AUC indicating better performance.

```
Accuracy :  81.03448275862068
Report :
              precision    recall  f1-score   support

           0       0.85      0.94      0.90        50
           1       0.00      0.00      0.00         8

    accuracy                           0.81        58
   macro avg       0.43      0.47      0.45        58
weighted avg       0.74      0.81      0.77        58
```

Figure 3.10: OpCode DecisionTreeModelTraining with Accuracy

Model Tuning and Selection: The hyperparameters of the decision tree model were tuned to improve the model's performance. This was done by varying the maximum depth of the tree, the minimum number of samples required to split a node, and the criterion for splitting a node. The model with the best performance on the test set was selected as the final model. Overall, this process was repeated for each feature set (i.e., text mining and opcode analysis) to create and evaluate decision tree models for the project.

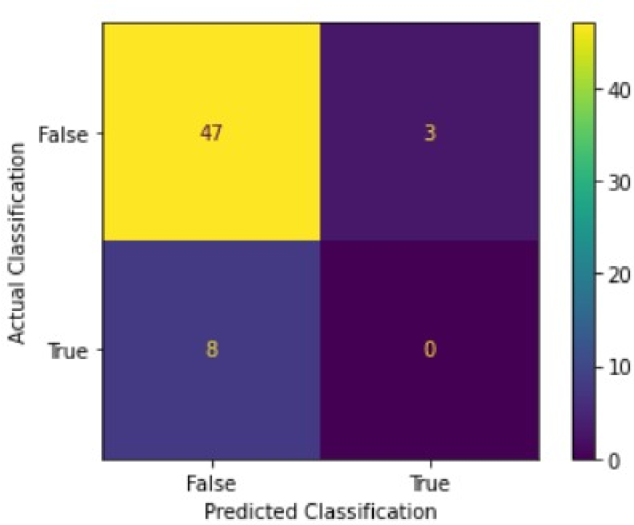


Figure 3.11: Confusion Matrix for average Accuracy in OpCodeAnalysis

The OpCode Decision Tree Model was run over multiple random states for various accuracies as shown(Accuracy, Random state), [86.20, 0][81.03, 10] [75.86, 20][82.75, 30][81.03, 40][75.86, 50][82.75, 60][86.20, 70][82.75, 80][72.41, 90][84.48,100]

The average Accuracy was calculated to be 81.029%

The AUC-ROC Curve for OpCode Analysis is as shown below.

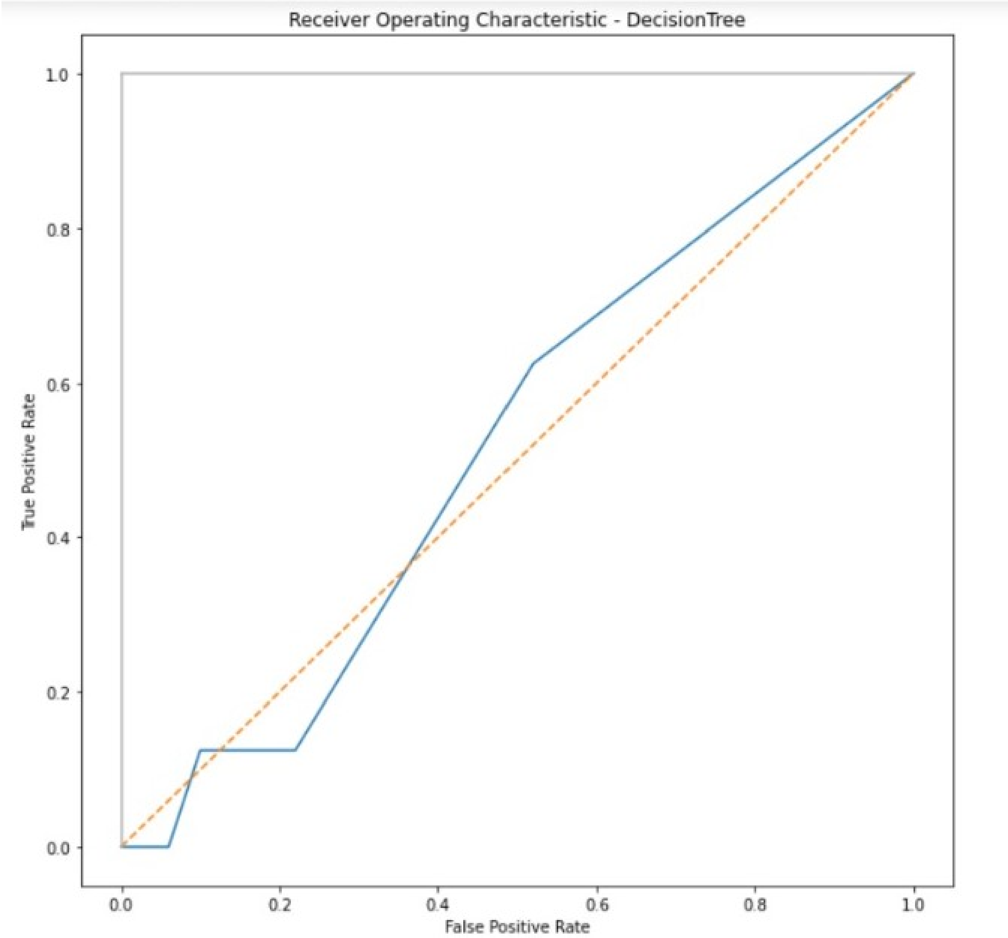


Figure 3.12: AUC-ROC Curve for OpCode 2

3.3.4 Text Mining Research Analysis

Similar keywords were observed among fake apps, and among genuine apps. These were carefully listed. A list of Keywords was manually created through and saved in .txt file. A program used to identify opcodes in a file and store their count value in a csv file was created.

Table 3.3: Text Mining Keyword used for dataset extraction

Telephony	Connection()	GetDeclaredMethod
Permission.ACCESS	Permission.Camera	registerReceiver
Permission.RECORD	BOOT_COMPLETED	ZipInputSTream
RequestPermissions	StartActivityForResult	
GetPackageInfo	GetInstallledApplication	

Keywords like telephony and connection() was observed to be used more often in fake apps compared to original ones, as they have to constantly connect and send stolen data to remote servers

BOOT COMPLETED keyword is used by apps to run in background without user knowledge once permission granted, fake apps use these oppurtunities to steal user data without their knowledge so some common use is to constantly ask permission for background use until given then silently steal data getInstalledPackages keyword is used to get information list of all the installed applications which very less legitimate apps ask permission for but fake apps constantly asks its permission

The dataset generated through Text Mining is used as a Training and Testing dataset for Text Mining Accuracy Analysis as shown.

- **Result and Accuracy Analysis**

  Model Evaluation: To evaluate the performance of the decision tree model, a confusion matrix was created. The confusion matrix shows the number of true positives, false positives, true negatives, and false negatives for a given classification model. The accuracy, precision, recall, and F1-score were calculated from the confusion matrix to evaluate the model's performance.

```
Confusion Matrix:  [[52  0]
 [ 6  0]]
Accuracy :  89.65517241379311
Report :               precision    recall  f1-score   support

           0       0.90      1.00      0.95        52
           1       0.00      0.00      0.00         6

    accuracy                           0.90        58
   macro avg       0.45      0.50      0.47        58
weighted avg       0.80      0.90      0.85        58
```

Figure 3.13: Text Mining Decision Tree Training and Accuracy Analysis

Additionally, an AUC-ROC curve was created to evaluate the model's performance. The AUC-ROC curve shows the true positive rate (sensitivity) against the false positive rate (1-specificity) for different classification thresholds. The area under the curve (AUC) represents the overall performance of the model, with a higher AUC indicating better performance.

Model Tuning and Selection: The hyperparameters of the decision tree model were tuned to improve the model's performance. This was done by varying the maximum depth of the tree, the minimum number of samples required to split a node, and the criterion for splitting a node. The model with the best performance on the test set was selected as the final model. Overall, this process was repeated for each feature set (i.e., text mining and opcode analysis) to create and evaluate decision tree models for the project.

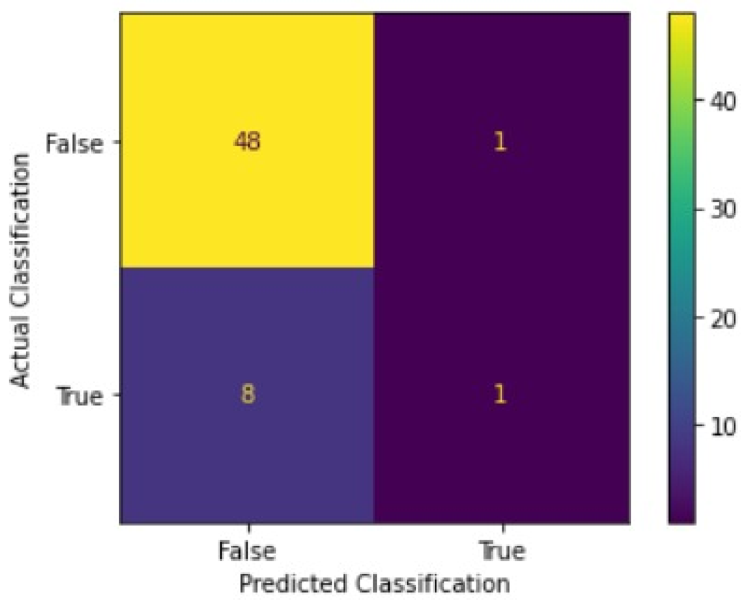


Figure 3.14: Confusion Matrix for average Accuracy in Text Mining

The Decision Tree Model Generated underwent multiple random states of dataset distribution to arrive at the final average acuuracy of 84.047%
The achieved accuracy has imporoved in Text Mining by approximately 3% from Opcode analysis.

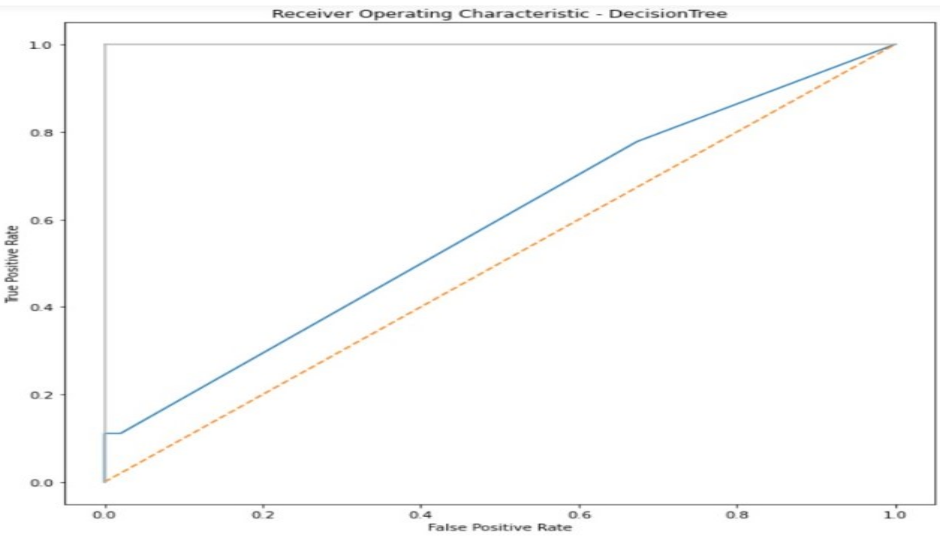


Figure 3.15: AUC-ROC Curve for Text Mining

3.4 Technologies used:

The proposed project utilizes several technologies to achieve its objectives. These include:

Kali Linux: Kali Linux is a Linux-based operating system that is specifically designed for penetration testing and digital forensics. In the proposed project, Kali Linux is used to convert APK files to Smali files and extract relevant data. This is done using tools such as Apktool, which is a tool for reverse engineering Android apps, and Dex2jar, which is a tool for converting Dex files (which contain compiled Android app code) to Jar files (which can be decompiled into Java source code). Once the APK files have been converted to Smali files, scripts written in Java can be used to extract relevant data, such as Dalvik opcodes.

Java: Java is a popular programming language used to develop Android apps. In the proposed project, Java is used to write scripts that perform pre-processing tasks, such as extracting Dalvik opcodes from Smali files. Java is also used to perform other tasks, such as parsing data files and generating reports.

Python: Python is a popular programming language used for data analysis and machine learning. In the proposed project, Python is used to develop the decision tree and random forest algorithms. Python is also used for data pre-processing, such as cleaning and formatting the data, and for data analysis, such as visualizing the data.

Scikit-learn: Scikit-learn is a machine learning library for Python that provides tools for data mining and data analysis. In the proposed project, Scikit-learn is used to train the decision tree and random forest models. Scikit-learn provides a range of machine learning algorithms, such as decision trees, random forests, and support vector machines (SVMs), as well as tools for model selection and evaluation.

Web scraping tools: Web scraping is the process of extracting data from websites. In the proposed project, web scraping tools such as BeautifulSoup and Scrapy are used to collect meta data and categories of apps from various sources. BeautifulSoup is a Python library for parsing HTML and XML documents, while Scrapy is a Python framework for web scraping.

Pandas: Pandas is a Python library for data manipulation and analysis. It provides tools for reading and writing data to a variety of file formats, such as CSV, Excel, and SQL databases. In the proposed project, Pandas is used to load, manipulate, and clean the data before it is used for machine learning. For example, Pandas can be used to remove duplicates, fill in missing values, and encode categorical variables.

Anaconda: Anaconda is a distribution of Python and R programming languages for scientific computing and data science. It comes with a range of pre-installed libraries and tools, including Pandas, Scikit-learn, and Jupyter notebooks. In the proposed project, Anaconda can be used to create a virtual environment for the project that includes all the necessary libraries and tools. This can help ensure that the project runs smoothly and that there are no version conflicts between different libraries.

Overall, the proposed project utilizes a range of technologies, including Kali Linux, Java, Python, Scikit-learn, web scraping tools, Pandas, and Anaconda, to achieve its objectives. By combining these technologies, the project aims to develop an effective and efficient system for classifying clone apps and enhancing mobile device security.

Chapter 4

Conclusion Future Works

4.1 Conclusion

In this proposed project, we have presented a methodology for classifying clone apps using reverse engineering and machine learning techniques. The proposed methodology involves generating Smali code from APK files, pre-processing the data using Dalvik opcodes and frequent malicious keyword analysis, and performing web scraping of apps metadata and categories. After feature selection, decision tree algorithms followed by random forest are applied to classify the apps as benign or malicious.

The importance of clone app detection cannot be overstated as they pose a serious threat to mobile device security. With the exponential growth in the number of mobile apps, the risk of clone apps has also increased, and it has become a major challenge for app stores and security professionals. The proposed project aims to address this challenge by providing an efficient and accurate system for detecting clone apps.

4.2 Future Works

The proposed project has several potential directions for future work, including:

Improving the accuracy of the classification model: The proposed project uses decision tree algorithms followed by random forest for classification. However, there are other machine learning algorithms that could be explored for improving the accuracy of the classification model. For example, deep learning techniques such as convolutional neural networks (CNNs) could be applied to the image analysis of apps icons.

Integration with existing mobile device security solutions: The proposed project can be integrated with existing mobile device security solutions to enhance their capabilities. For example, the classification model could be integrated with antivirus software to detect and remove clone apps from mobile devices.

Real-time clone app detection: The proposed project is designed to classify clone apps offline. However, it is also possible to develop a real-time clone app detection system that can detect and remove clone apps in real-time as they are installed on mobile devices.

Extending the dataset: The proposed project uses a dataset of clone apps collected from various sources. However, the dataset can be extended by including more samples from different app stores and sources to increase the diversity of the data and improve the performance of the classification model.

In conclusion, the proposed project presents a promising approach to clone app detection using reverse engineering and machine learning techniques. The project has significant potential for improving mobile device security and addressing the growing threat of clone apps.

References

[1] Singh, A., Narula, R., Kumar, R. (2017). CloneDetective: Detecting Repackaged Smartphone Applications Using Cloned Code Detection Techniques. Journal of Digital Forensics, Security and Law, 12(3), 33-52.

[2] Zimmeck, S., Lindorfer, M., Mulliner, C., Weichselbaum, L. (2015). Who Are You? Detecting Smartphone Users with Sensor-Enhanced Smartphones Using Machine Learning. International Journal of Information Security, 14(5), 443-456.

[3] Ma, T., Xie, C., Wu, L., Chen, H. (2022). DeepClone: Detection of Repackaged Smartphone Applications Using Deep Learning Techniques. Journal of Information Security and Applications, 67, 102889.

[4] `https://9apps.onl/9apps-download/`

[5] `https://apk.support/search?q=downloader`

[6] `http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html`

[7] `https://apkgk.com/APK-Downloader`